BERGER Jordan

https://www.codingame.com/profile/c6231376e80897d910854f3a03b9e8117380273
https://github.com/joberger95/CoderStrikesback_Python

Suivi de programme

Passage de la Ligue BOIS à la ligue BRONZE.

```
    import sys
    import math

    while True:

        x, y, next_checkpoint_x, next_checkpoint_y, next_checkpoint_dist,
next_checkpoint_angle = [int(i) for i in input().split()]
        opponent_x, opponent_y = [int(i) for i in input().split()]

        if next_checkpoint_angle > 90 or next_checkpoint_angle < -90:
            thrust = 0
        else:
            if next_checkpoint_dist > 2000:
                thrust = "BOOST"
            else:
                thrust = 100

        print(str(next_checkpoint_x) + " " + str(next_checkpoint_y) + " " +
str(thrust))
```

13h00

Il est prévu que je passe à la ligue suivante dans 35 minutes, soit la Ligue ARGENT. Je n'ai fait qu'une très légère modification. On remarque aussi le morceau en commentaire qui avait pour lieu d'être un test. Le programme actif est celui ci-dessous.

```
import sys
import math

while True:

    x, y, next_checkpoint_x, next_checkpoint_y, next_checkpoint_dist,
next_checkpoint_angle = [int(i) for i in input().split()]
    opponent_x, opponent_y = [int(i) for i in input().split()]

    if next_checkpoint_angle > 90 or next_checkpoint_angle < -90:
        thrust = 0
    else:
        if next_checkpoint_dist > 4000:# and opponent_y == 400 and opponent_x ==
```

```
400:
            thrust = "BOOST"
        else:
            thrust = 100

    print(str(next_checkpoint_x) + " " + str(next_checkpoint_y) + " " +
str(thrust))
```

```
import sys
import math

#Debug tool (from V.POULALLEAU)
def debug(*args, **kwargs):
    print(*args, **kwargs, file=sys.stderr)

#Opponent proximity
def opponent_proximity( x , y , opponent_x , opponent_y ):
    pod = ( opponent_x - x )**2
    pod_opponent = ( opponent_y - y )**2
    square_of_sum = math.sqrt(pod + pod_opponent)
    debug(square_of_sum)
    return square_of_sum
# Auto-generated code below aims at helping you parse
# the standard input according to the problem statement.

#Globals variables
available_boost = True
# game loop
while True:
    # next_checkpoint_x: x position of the next check point
    # next_checkpoint_y: y position of the next check point
    # next_checkpoint_dist: distance to the next checkpoint
    # next_checkpoint_angle: angle between your pod orientation and the direction
of the next checkpoint
    x, y, next_checkpoint_x, next_checkpoint_y, next_checkpoint_dist,
next_checkpoint_angle = [int(i) for i in input().split()]
    opponent_x, opponent_y = [int(i) for i in input().split()]

    # Write an action using print
    # To debug: print("Debug messages...", file=sys.stderr)

    # You have to output the target position
    # followed by the power (0 <= thrust <= 100) or "BOOST"
    # i.e.: "x y thrust
    if next_checkpoint_angle <= 90 or next_checkpoint_angle >= -90:
        if next_checkpoint_dist > 1000:
            thrust = 100
        if next_checkpoint_dist > 750 and next_checkpoint_dist <= 1000:
            thrust = 50
        if next_checkpoint_dist < 250:
            thrust = 25
```

```
    if next_checkpoint_angle > 90 or next_checkpoint_angle < -90:
        thrust = 15

    if next_checkpoint_dist > 1000 and  available_boost == True:
        thrust = "BOOST"
        available_boost = False

    pods_dist_diff=opponent_proximity(x , y , opponent_x , opponent_y)
    if pods_dist_diff <= 450:
        thrust = "SHIELD"

    print(str(next_checkpoint_x) + " " + str(next_checkpoint_y) + " " +
str(thrust))
    debug(thrust)
```

```
import sys
import math
import time

#Debug tool (from V.POULALLEAU)
def debug(*args, **kwargs):
    print(*args, **kwargs, file=sys.stderr)

def distance_calculation( coo_x1 , coo_x2 , coo_y1 , coo_y2 ):
    return math.hypot(abs(coo_x2 - coo_x1), abs(coo_y2 - coo_y1))

# Auto-generated code below aims at helping you parse
# the standard input according to the problem statement.

#Globals variables
available_boost = True
# game loop
while True:
    # next_checkpoint_x: x position of the next check point
    # next_checkpoint_y: y position of the next check point
    # next_checkpoint_dist: distance to the next checkpoint
    # next_checkpoint_angle: angle between your pod orientation and the direction
of the next checkpoint
    x, y, next_checkpoint_x, next_checkpoint_y, next_checkpoint_dist,
next_checkpoint_angle = [int(i) for i in input().split()]
    opponent_x, opponent_y = [int(i) for i in input().split()]

    # Write an action using print
    # To debug: print("Debug messages...", file=sys.stderr)

    # You have to output the target position
    # followed by the power (0 <= thrust <= 100) or "BOOST"
    # i.e.: "x y thrust
    if next_checkpoint_angle <= 90 or next_checkpoint_angle >= -90:
        if next_checkpoint_dist > 1000:
```

```
            thrust = 100
        if next_checkpoint_dist > 750 and next_checkpoint_dist <= 1000:
            thrust = 50
        if next_checkpoint_dist < 250:
            thrust = 25

    if next_checkpoint_angle > 90 or next_checkpoint_angle < -90:
        thrust = 15

    if next_checkpoint_dist > 1000 and 25 <= next_checkpoint_angle <= -25 and
available_boost == True:
        thrust = "BOOST"
        available_boost = False
    else:
        thrust = 100

    pods_dist_diff = distance_calculation(coo_x1 = x , coo_x2 = opponent_x ,
coo_y1 = y , coo_y2 = opponent_y)
    if pods_dist_diff <= 1500 and next_checkpoint_dist < 500:
       thrust = "SHIELD"

    print(str(next_checkpoint_x) + " " + str(next_checkpoint_y) + " " +
str(thrust))
    debug(thrust)
```

Difficultés

Suivi de code

Par simple manque d'assiduité et de rigueur, j'ai tout simplement omis d'effectuer un suivi correct du programme. L'erreur qui en découle est similaire à ce qu'on peut faire aujourd'hui avec un gestionnaire de version tel que GitHub. Cela contre-carre en effet ce procédé et permet de revenir aisément à des solutions passsées qui fonctionnaient totalement ou partiellement. Ce qui n'a pas été réalisé tout au long du projet et ne me permet pas donc de montrer mes nombreuses recherches et/ou idées.

Traduction Homme-machine

L'objectif est de coder une IA de course. il m'a semblé difficile, je n'ai d'ailleurs pas réussi, de mettre en oeuvre les idées, mouvements, principes qui me paraissent naturels en tant qu'Homme et de la traduire via un programme pour retrouver des comportements similaires. Il ne m'a pas été possible de traduire des notions telles que l'instinct, par exemple. Ce qui est une tare en terme de stratégie pour le pods. Je n'ai pas non plus réussi à traduire une forme de trajectoire, nottament par manque de compétences mathématiques. *cf : fichier github, pb_Maths.pdf*

Le "manque" de données

J'ai toujours eu le ressentit que récupérer tout ou parti de projet me met davantage en difficulté. Je ressens constamment un manque d'informations, de possibilités d'utilisations de données. La programmation pour la

course ne propose que très peu de choses à dispositions et les modules ou variables supplémentaires ne se débloquent qu'en progressant dans les Ligues. Il n'en reste pas moins intéressant de travailler avec ce niveau de difficulté pour progresser et apprendre.

Point(s) d'attention

Pas de points à regarder particulièrement. J'ai tenté de mettre en oeuvre mes idées et les notions vues en cours.

Retour d'expérience

Cet exercice était très intéressant pour appréhender l'univers de l'intelligence artificielle et ses difficultés, *cf Traduction Homme-machine*. Le langage de programmation Python est vraiment puissant et intelligemment construit, ce qui permet à tout programmeur quelque soit son niveau et ses objectifs d'y trouver ou retrouver un plaisir de codage et une volonté d'avancer. C'est tout du moins ce que j'ai ressentit, bien que n'ayant au premier abord pas un attrait important pour cette discipline.